Knowledge Based Techniques to Increase the Flexibility of Workflow Management[1]

Barbara Dellen, Frank Maurer, Gerhard Pews

AG Expertensysteme, Universität Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern
e-Mail: {dellen, maurer, pews}@informatik.uni-kl.de
http://wwwagr.informatik.uni-kl.de/~comokit

Keywords
Ad-hoc workflow, flexible workflows, traceability, project coordination, dependency management

Abstract

This paper describes how knowledge-based techniques can be used to overcome problems of workflow management in engineering applications. Using explicit process and product models as a basis for a workflow interpreter allows to alternate planning and execution steps, resulting in an increased flexibility of project coordination and enactment. To gain the full advantages of this flexibility, change processes have to be supported by the system. These require an improved traceability of decisions and have to be based on dependency management and change notification mechanisms. Our methods and techniques are illustrated by two applications: Urban land-use planning and software process modeling.

1 Motivation

On account of global competition, the efficiency of business processes has to be improved, resulting in a reduction of process requirements in terms of time and cost. This business objective leads to approaches as *lean management* and *business process reengineering & optimization*. In order to fulfill these objectives, workflow management approaches often are introduced in the enterprise. The workflow management coalition defines workflow management as:

> „The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.“ [40]

Galler, Hagemeyer and Scheer [13] describe the life cycle of workflows (see Figure 1). The development of a workflow application starts with analyzing and modeling the business process. The business process model is the input for the workflow system development which results in a workflow model, typically implemented in an workflow language. The resulting application is repetitively executed within the company, hopefully improving and supporting business activities. Storing traces of the workflow execution and analyzing it is a basis for future improvements of the process (thereby supporting an on-going improvement process/Kaizen [30]). The authors state:

> „The transformation of business concepts (expressed in Business Process Models) into information systems (for example WMS) is not a simple derivation but a creative process which includes many feedback loops between organizational and technical experts.“ [13]

Their line of argument is that there is a (huge) human effort in mapping the abstract business model onto an operational and executable workflow language. Because of the necessary human effort, there are development costs which cannot be neglected. Hence, this approach basically is only applicable for

1. To appear in: Data & Knowledge Engineering Journal, North-Holland, 1997

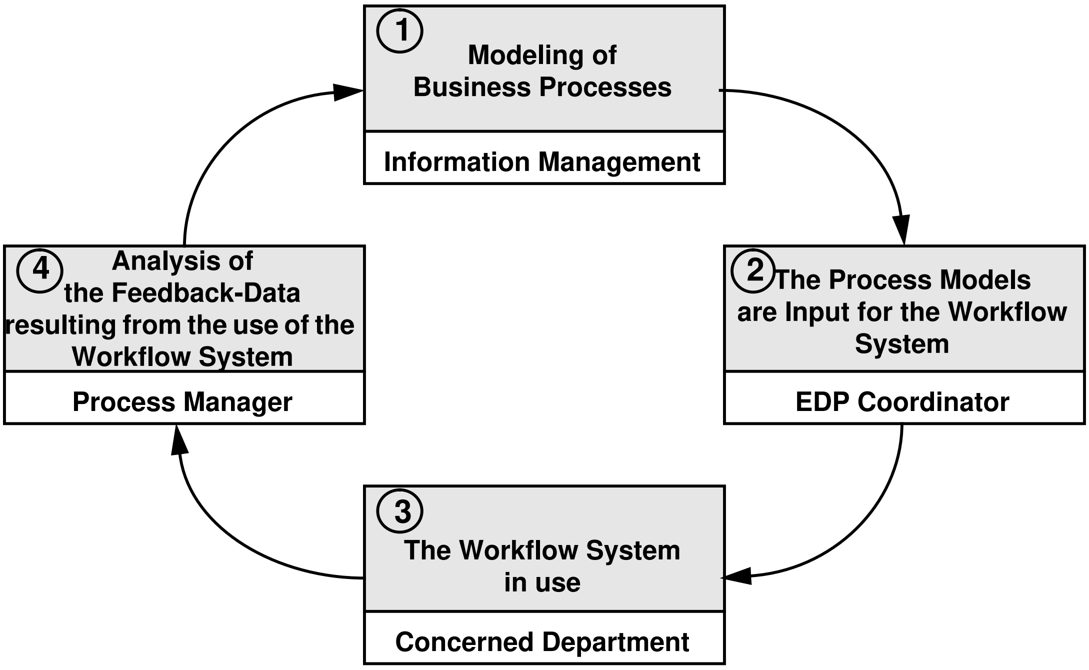

Figure 1 Workflow Life Cycle (from [13])

repetitive, standardizable processes as, for example, administrative activities like the payment for traveling costs.

Additionally, it should be mentioned that Galler et al. assume a *two-phase approach* where the modeling phase must be completed before the execution of the workflow may start. Although there is a tremendous profit in supporting administrative workflows following a two-phase approach, not all processes fall into this category. Galler and Scheer distinguish in [14] between case-oriented workflows and ad-hoc workflows. The former follow some rules but are not completely standardizable, whereas the latter are dealing with completely unstructured single-instance processes. Georgakopolous and Hornick gave a similar definition of ad-hoc workflows:

> „Ad-hoc workflows perform processes, where there is no set pattern for moving information among people. Ad-hoc workflows typically involve human coordination, collaboration or co-decision. Thus the ordering and coordination of tasks (activities) in an ad-hoc workflow are not automated but are instead controlled by human. The task ordering and coordination decisions are made while the workflow is performed.“ [16]

To support ad-hoc workflows, often groupware approaches are proposed. Groupware systems (e.g. Lotus Notes) basically allow to share information between the group members. Compared to workflow approaches this has some drawbacks:

- Handling worklists of users is not supported. Hence, the system is not able to proactively send messages to users about activities to be carried out. The system does not guide its users in their work.
- Tracing task execution is not possible because the system has no notion of „Tasks“.

Nevertheless, groupware allows an increased flexibility of the work processes. This flexibility is needed for design, engineering or science which are important application domains. Wainer et al. argue in the same direction:

> „In real life, both in office and in scientific lab environments, the enactment of a workcase may

deviate significantly from what was planned/modeled.“ [41]

To summarize, groupware systems provide too little structure and guidance whereas current workflow techniques are not flexible enough to support virtual corporations, except in a very limited way.

A more flexible approach than current workflow techniques, without the drawbacks of groupware is particularly needed in distributed engineering projects where many of the tasks cannot be analyzed prior to execution, and where the duration of co-operation is typically very long. In a sense engineering projects are the most challenging case of virtual companies that may in general be very dynamic. Distributed engineering projects are also particularly interesting because of the obvious need to reduce cost and increase speed by allowing engineers and designers in different organizations to work directly in a peer-to-peer relationship rather than being separated by several layers of management. This would enable an enterprise to sell its products earlier and cheaper than its competitors.

Our applications come from the following engineering domains: urban land-use planning and software engineering. Looking from this perspective leads to the wish to combine the advantages of workflow and groupware approaches, overcoming their drawbacks. In the rest of this paper, we will show how knowledge-based techniques can be used to reach this goal.

The next section states requirements which must be fulfilled by a workflow tool in engineering domains. Chapter 3 gives an overview over our application. The Software Engineering application is used to illustrate the technical details of our approach which are described in Chapter 4 - Chapter 7. In Chapter 8, we discuss the implementation of our system. Related work is discussed in Chapter 9. The last section summarizes our results and future work.

2 Global Goals and Global Answers

The global goal of our work is to develop methods, techniques and software systems which support the planning and execution of large-scale engineering projects. Although our approach mostly deals with engineering, we assume that our techniques are applicable in all domains where flexible work processes have to be supported.

A closer look on our engineering applications showed that they can be characterized by the following features:

- People work in different locations and at different times on a shared task. Time and money are spend for planning and coordination of the work process. The plan can not be modeled on a fine-grained level before the execution starts. Therefore, the two-phase workflow approach is not applicable for them. Basically, the project plan must be seen as an equivalent to a process model.
- Large-scale development processes are long-term processes. Requirements on the outcome may and will, as a fact of life, change during process execution. Therefore, the need to revise and adapt the project plan and the results on the fly is inherent in these processes. Change processes are a central reason for exceeding the project schedule and budget.
- During the process, a lot of decisions are made by each participant. These decisions influence each other and base on each other. Many of these decisions are made on unstable ground: required information may be missing when the decision is made. Hence, decisions may change over time, raising the need for an automatic change management and propagation mechanism.

These features lead to certain requirements for process-support tools which are in the focus of our work:

- The tool has to support the planning of the work process. The resulting plan must be mapped *automatically* onto an operational workflow model because an explicit implementation phase (as in two-phase workflow approaches) would be too costly and too time consuming.
- Spatial and chronological separation of the tasks necessitates that the tool serves as a kind of "intelligent memory" for its users, in order to allow participants to understand the rationales behind decisions and to access relevant informations.
- If changes take place, the tool should automatically inform the appropriate users. To determine the set of users who are affected by a change, the tool has to incorporate a notion of dependencies between information and a generic and/or domain-specific theory for their generation [27].

2.1 Our Answers - An Overview

To address the mentioned problems, we integrate knowledge-based techniques with a workflow management approach. Here we give an overview over our approach. The next section gives more details.

Explicit process & product models: Based on knowledge and software engineering approaches [5, 6, 38], we developed an ontology which allows to describe work processes. The basic notions of this ontology are process models, methods, product models, parameters, and agents. Using these concepts, users are able to define a project plan.

A workflow engine which interprets the process models: The basic approach of knowledge-based systems, is to distinguish between the knowledge about a domain and the interpreter which uses it. We adopted this approach and developed a workflow engine which uses explicit process & product models as input to support the execution of the project. The engine supports project coordination, e.g. by providing worklists for its users, and is used to exchange information between team members.

Generate dependencies between information based on the process model: Since change management is a must for engineering project support, we built a generic dependency theory into our workflow engine which automatically extracts dependencies from the process model. Based on these dependencies, our system is able to proactively send notifications to users who are effected by a change.

Allow to refine and extent the process model on the fly: Our system allows to alternate between planning and execution steps. Because of the interpretative workflow engine process models can be extended during execution. Furthermore, we provide the concept of a method within our ontology. A method describes a way to reach the process goal. This concept enables a project planner to select a solution strategy for a process during execution.

Allow to change planning decisions: Since the workflow engine „sees“ the current project plan only as data and not as program code, it is possible to change the plan on the fly. The change propagation mechanisms are able to inform appropriate users about such a change. This leads, for example, to updated worklists or new inputs for a task.

In the next section, we give an overview over our applications. In the following these will be used as examples for our approach.

3 Applications

Our approach is applied to two design domains: Urban Land-Use Planning and Software Engineering, which will be introduced in this section.

3.1 Urban Land-Use Planning

In Germany, municipalities are responsible for determining and controlling the future development of the cities. One step in doing this is to work out a legally binding land-use plan. This plan consists of textual stipulations and stipulations by drawing, which determine the type and degree of building and land use for areas within the municipality. For example, they might state that a certain area has to be a residential-only area (type of building and land use), or specify the maximum amount of storeys allowed and the heights of physical structures (degree of building and land use). Legally binding land-use planning is part of local legislation and therefore there are certain requirements for plan development as well as for the completed plan. The procedure of setting up a plan has to be traceable, and consistent with the law.

For this domain, we used our techniques to build a system which gives intelligent support to an urban planner. The IBP-System (Intelligenter Bebauungsplan - Intelligent Land-Use Plan) focuses on the process of transforming a first design study into the legally binding plan [28]. Typically, it will be performed by a single person at a time, but with large temporal gaps in it. Setting up a plan might take several years time, during which participants and planners might change. Therefore it is extremely important for anybody working with this plan to keep in mind the dependencies and rationales for all stipulations. Moreover, there is a great benefit from pre-structuring the urban planning process, in order to make the process more efficient and not to neglect any legal decrees or standards. After the plan has been completed, its intentions should be communicated to interested citizens or investors.

The IBP-Project is an interdisciplinary research project between three research groups in Computer Science, Environmental Planning and Jurisprudence[2]. In two years‘ work, we set up a detailed model of the design process and the involved data. Basing on this, we built a support system for land-use planning which keeps track of the planning process and provides information filtering and tool (CAD, Geographic Information System/GIS) configuration for these planning tasks.

3.2 Process centered software development[3]

In Software Engineering, the need for process engineering has become obvious [21, 17]. In order to improve the software process and the software, process modeling, project planning and management, change control, quality measurement and reuse [22, 36] have to be introduced. For this, several process centered software development systems to guide, reason about, control, and coordinate software processes [12, 15, 39, 36] have been developed. Two characteristics of software processes that influence research in this area are their long duration where necessary changes occur, and the large

2. The project is supported by the Volkswagen-Stiftung. Involved are in Computer Science: Prof. Dr. Michael M. Richter’s group “Expert Systems / Artificial Intelligence”, in Jurisprudence: Prof. Dr. Rudolf Stich’s group “Science of Law and Administration”, and in Town and Environmental Planning, the group “CAAD and Planning Methods” of Prof. Dr.-Ing. Bernd Streich.

3. This work is supported by the *Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 501, Teilprojekt B2.*

contingent of creative processes, which cannot be automated. Therefore, computer based project planning and enactment support has to be flexible and to leave room for situation dependent decisions by the user. The methods and techniques of our approach provide solutions for computer based software process support, with respect to both characteristics. Within the MILOS [39] project, we implement a process centered software development environment. A future goal of the project is to integrate software measurement and the reuse of products, measures, and process models in our approach.

4 CoMo-Kit System Architecture

To build our applications, we developed the CoMo-Kit system which defines and implements an ontology for project planning and incorporates an interpreter for its operationalization.

Figure 2 shows the basic system architecture of CoMo-Kit [28], [29]. It consists of two main parts:
- The *Modeler* defines and implements an ontology which can be used for project planning. Using the Modeler, team members are able to plan a complex design project.
- The *Scheduler* operationalizes the project plan. It supports the execution of a project and manages the information produced.

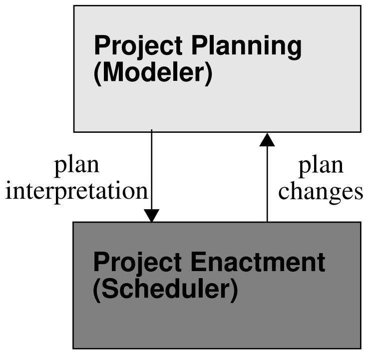

Figure 2 Architecture of the CoMo-Kit System

For a new development project, in a first step an *initial project plan* is created using the Modeler. For our application domains, we developed generic project models which are initial project plans.

This plan contains descriptions of the processes to be done, a definition of the product data structures which must be created during plan execution and a list of the team members involved in the design process. The next section explains this in further detail.

The current project plan is used by our Workflow Management System, the CoMo-Kit Scheduler, to support project execution. The Scheduler interprets the plan information. It supports team members in their work by
- generating worklists,
- providing access to relevant information for executing a process,
- forwarding results of a process to other team members,
- allowing to delegate processes to other team members,
- supporting the supervision and management of delegated processes,
- notifying the appropriate persons about changes within the project, and

• enabling further project planning using already available results of the project.

Now we explain the ontology of the Modeler.

5 Project Planning: The Ontology of the CoMo-Kit Modeler

Project planning can be seen as developing a model how the project should be carried out. To describe cooperative development processes, our Modeler uses an ontology consisting of four basic notions: Process Models, Methods, Product Models and Resources. The term „process model" comes from software process modeling. We use the term „task" synonymously. In the following, these terms are defined as far as is necessary to understand this paper, omitting the syntactical details of our project planning language. These details are also hidden from the user by a graphical user interface.

5.1 Process models

A process is characterized by a set of activities (e.g. delegation, product creation) to be executed in order to reach the process goal. Our application domains basically deal with the production of information. In contrast production processes have physical structures (cars, computers) as central output which cannot be physically stored in a computer system. Information which is relevant for the project is described in the project plan. Therefore, we associate every process with a set of input and output parameters. If a parameter is modified in a process, it is modeled as input *and* output. In the project plan, we are only able to state which type of information is used or must be produced.

Example: The process „requirements engineering" of the software process modeling application uses an object of type „Informal problem description" as input and produces an object of type „Requirements document" as output.

For every input the flags mentioned in table 1 are defined. The flags are mutually exclusive.

Table 1: Parameter flags

Flag Name	Meaning
needed for planning	The input must be available before the planning of the process starts
needed for execution	The input is not needed for planning but it must be available before the execution starts.
optional	The input neither needed for planning nor for enactment (but it may be helpful to have).

Outputs of a process may be optional or required.

Additionally, for every process a precondition and a postcondition may be defined. Preconditions are, for example, used to check if the inputs fulfill given requirements. Postconditions are, for example, used to check if the output of a process has a desired quality.

Example: For a process „implement program modules" the precondition may be „all module specifications completed" and the postcondition may be „module complexity < LIMIT".

When processes are executed, *decisions* are made which result in *assignments* of values to the output parameters. *Our approach assumes that there is a causal dependency between the available inputs and the produced outputs of a process.* We work on the principle that during project planning only inputs are associated to a process which are relevant and necessary for reaching the goal. Limitations of this assumption and solution for the resulting problems are discussed in [10].

For every process, the planner may state criteria which must be fulfilled by agents to be allowed to work on the process during execution. For example for an „implementation process“ in the software engineering domain, an agent should have skills in *Smalltalk-80 programming* and belong to department *ZFE 153*.

5.2 Product models

To describe product models, an object-centered approach is used. We adopted the term „Product model“ from software process modeling approaches. Basically, the term describes data types. As usual, we distinguish between product classes and product instances. Product classes define shared features for a set of products. Classes can be structured in an inheritance hierarchy and include a set of slots to structure the product. Every slot has a type and cardinality. Types may be other product classes or basic types (e.g. SYMBOL, STRING, REAL,...). Using other product classes as the type of a slot creates complex object structures (part-of decomposition). A product instance is an instance of a product class. For sake of brevity, we will use the term “product” for “product instance”.

Figure 3 shows a part of the product class hierarchy and an product class editor of the urban land-use planning application.

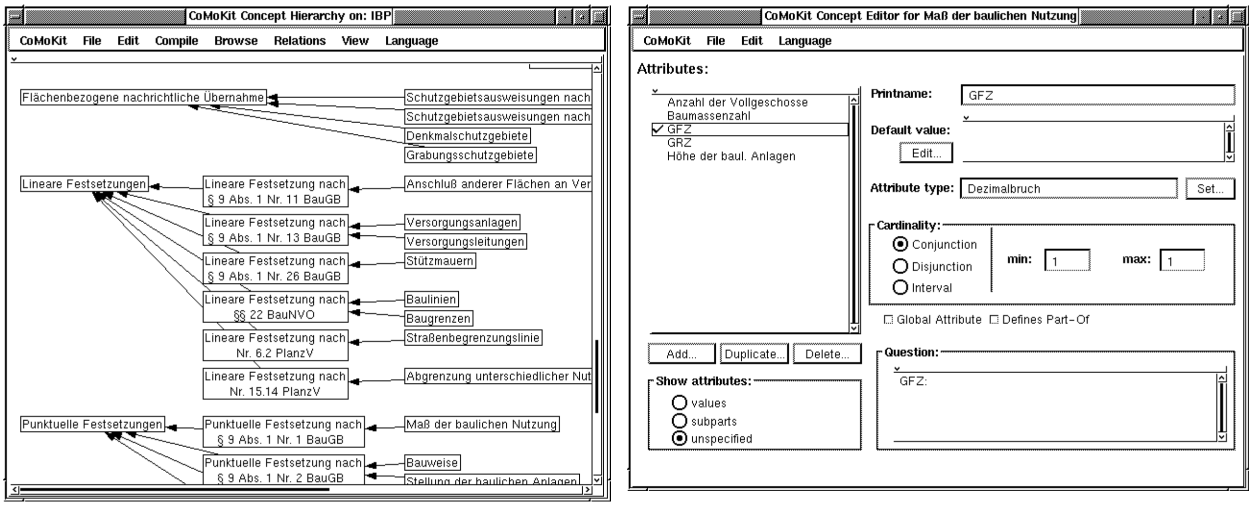

Figure 3 Product model definition tools

5.3 Methods

A method defines a solution strategy to reach the process goal. For every process, there may exist a set of alternative methods. This set might be extracted from old project traces and stored in an „experience factory“ [3]. To solve a process, a method has to be applied. Our approach allows furthermore to define new methods dynamically, i.e. during project planning and execution (see Chapter 7).

Methods are executed by agents (see below). Not every agent who will be responsible for a process may have the abilities to execute every method (For example, an „implementation process“ can be carried out by using the methods “Implement in C++” or “Implement in Smalltalk”. The planner must be able to specify that developer „Mr. Miller“ is able to implement the software in C++ but not in Smalltalk). Therefore, we allow to describe agent bindings for every method.

We distinguish between atomic and complex (or composed) methods.

- Atomic methods assign values (i.e. instances of product classes) to parameters. *Process scripts* describe for humans how a given process is to be solved. *Process programs* are specified in a formal language so that computers can solve a process automatically without human interaction.
- Complex methods describe the decomposition of a process into several subprocesses and define a product flow graph between the subprocesses. A *product flow graph* consists of nodes which define parameters & (sub)processes, and links which relate parameters to processes. The direction of the link determines if the parameter is an input or an output of the process. *We assume that there is a causal dependency between a process and its subprocesses* (see chapter 6.2).

Subprocesses can be further decomposed by methods, resulting in a process decomposition tree. Figure 4 shows an example process decomposition of the Software Engineering domain. One can see that there are two alternatives to solve the process „Implementation Process“, namely „Implementation with Structural Testing“ and „Implementation with Functional Testing“. Both methods decompose the process into several subprocesses. Method „Implementation with Structural Testing“, for example, defines five subprocesses: „Create Operating Documentation“, „Perform Integration“, „Create Test Data using Source Code“, „Create Source“, and „Generate Object Code“.

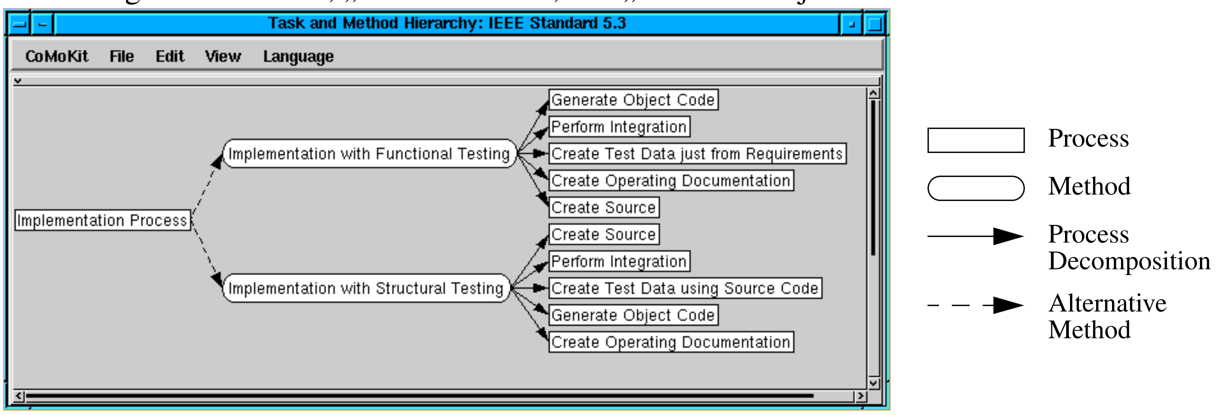

Figure 4 Process decomposition

In Figure 5 the product flow of method „Implementation with Structural Testing“ (see Figure 4) is given. One can see, that the „Design Document“ is used by the process „Create Source“. The resulting product is „Source Code“. For reasons of simplicity the type of the products is not displayed in the figure.

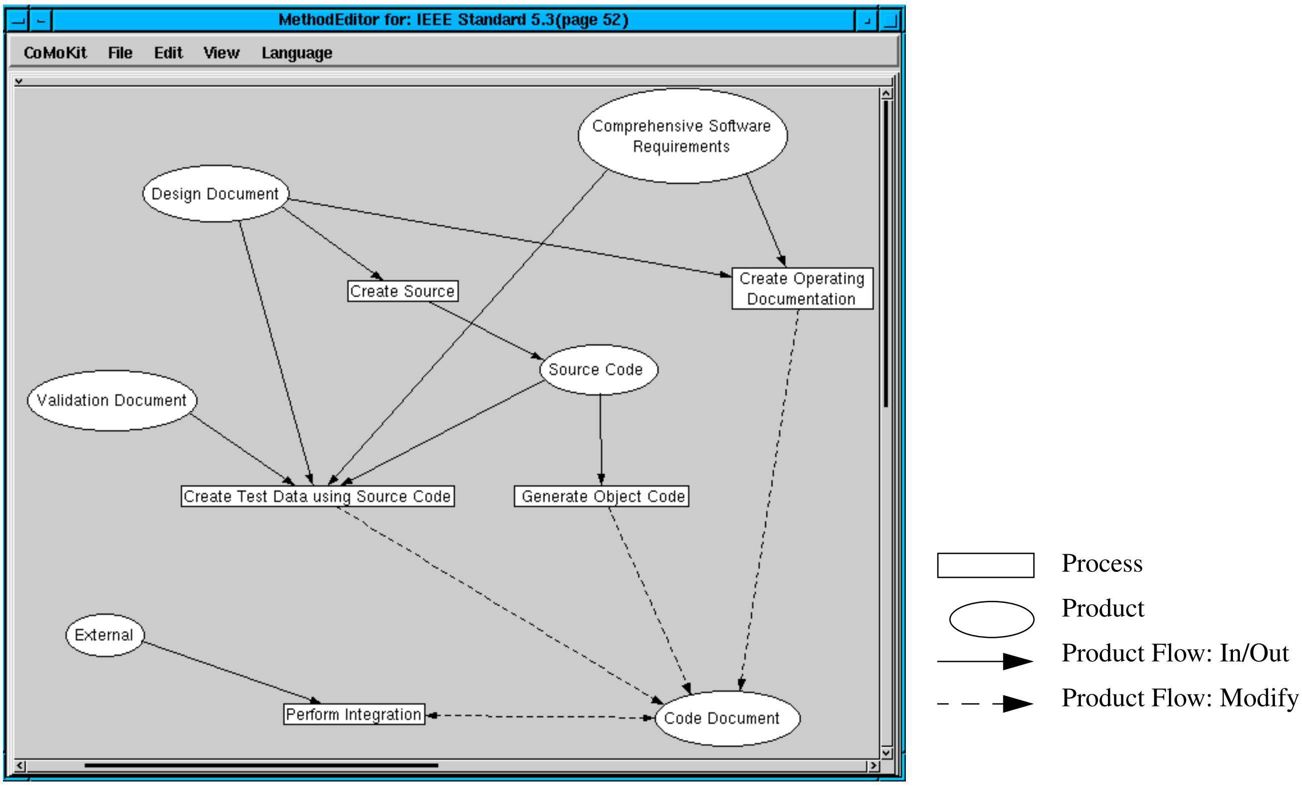

Figure 5 Product flow defini-

5.4 Resources: Agents & Tools

Resources are used for project planning and process execution. *Agents* are active entities which use (passive) *tools* for their work.

Processes are executed by agents. We distinguish between *actors* (i.e. human agents) and *machines*.

Our system stores information about the properties of every agent. For actors, we distinguish three kinds of properties: qualifications (q), roles (r), and organization (o).

During process execution, our system compares the properties required for working on a process with the properties the agents posses. This allows to compute the set of agents which are able to execute the process.

> *Example:* In a project plan, it is defined that the process „implement user interface“ should be executed by an actor which has skills in using the „Visualworks Interface Builder (q)“, is a „programmer (r)“, and works in „department Dep 1.4 (o)“.

Having sketched our ontology for project planning, we now will explain how the execution of plans is supported.

6 Project Execution

Our methods and techniques do not automate the project planning and execution. Instead, the project members are supported and guided in their activities. Information is exchanged interactively between project members and the Scheduler. The following scenario gives an impression what the execution support of our approach looks like on the user interface level. Sections 6.1 and 6.2 give more technical details of the system design. How we support change processes is described in Chapter 7.

Figure 6 shows a snapshot of the worklist of agent named „Barbara Dellen". The agent can switch

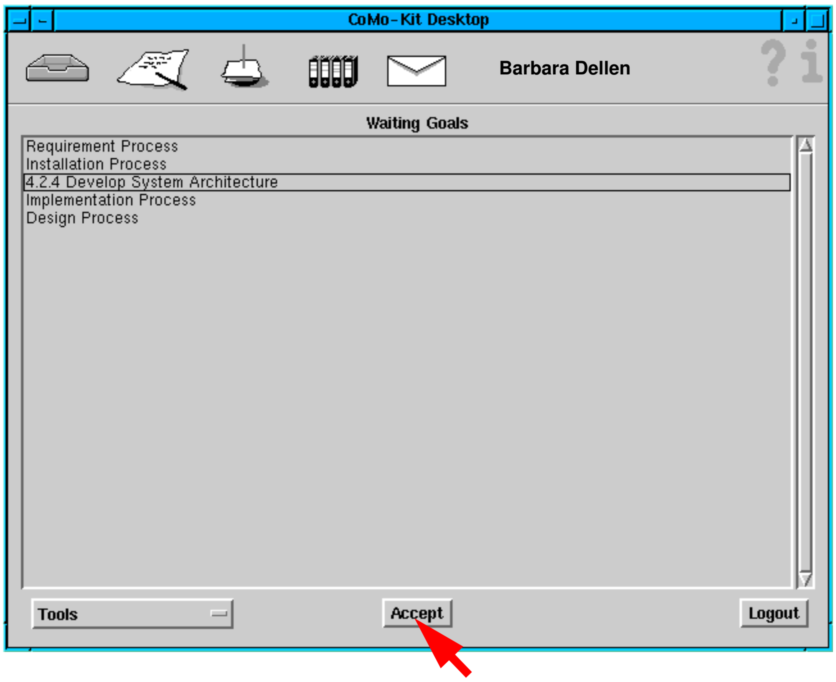

Figure 6 Worklist of an agent

between different agendas, in which the processes that are delegated to the agent are shown. Each agenda collects processes in different states: processes waiting for execution (symbol *letter tray)*, in progress (symbol *paper sheet)* or suspended (symbol *pin*). The waiting processes can be selected, planned or executed by the agent. In Figure 6 the agenda with the processes waiting for execution is displayed.

In this scenario, the agent „Barbara Dellen" decides to work on the process „4.2.4 Develop System Architecture", by pushing button „Accept" in Figure 6. This decision is propagated to and stored within the Scheduler. Because the Scheduler guides the agent as far as necessary in his/her activities, „Barbara Dellen" is requested to select a method (see Figure 7). The information about the available methods is extracted from the process model. After the agent selected one of them by pushing button „Select", the decision is returned to the Scheduler. As a result, the subprocesses defined by the method become part of the current project plan.

The next task of the agent is the delegation of processes to other agents. Figure 8 shows a window, where the processes actually to be delegated are shown in the upper left agenda. By pushing button

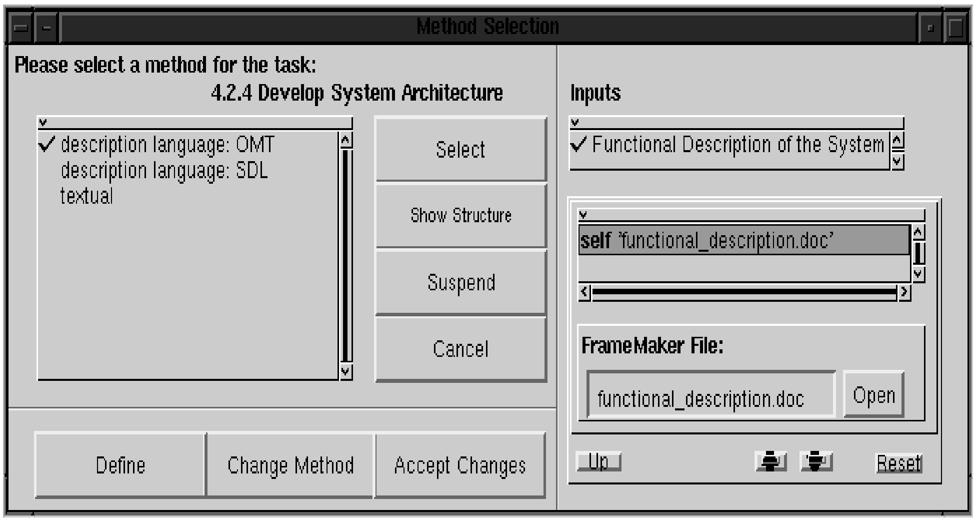

Figure 7 Method selection of process „4.2.4 Develop System Architecture“

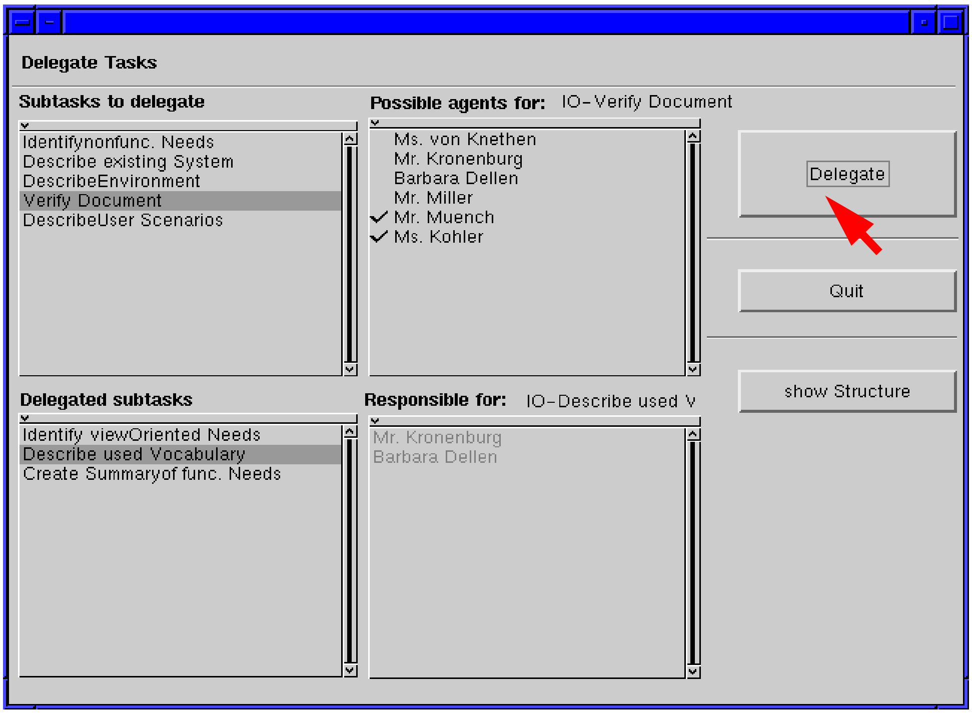

Figure 8 Delegating processes

„Delegate“, the process „Verify Document“ will be delegated to „Mr. Muench“ and „Ms. Kohler“. The agenda in the lower left side of the window displays the processes that have already been delegated. If one of them is marked, the lower right agenda displays the agents to whom the process has been delegated.

The last snapshot (Figure 9) shows the creation of a FrameMaker document. After an atomic method has been applied, the agent can have a look on the consumed products in the upper left agenda. In this case, the process „Analyze Requirements“ consumes two products. The products to be produced are shown in the upper right agenda. After selecting an output product (here: „System Architecture“) that

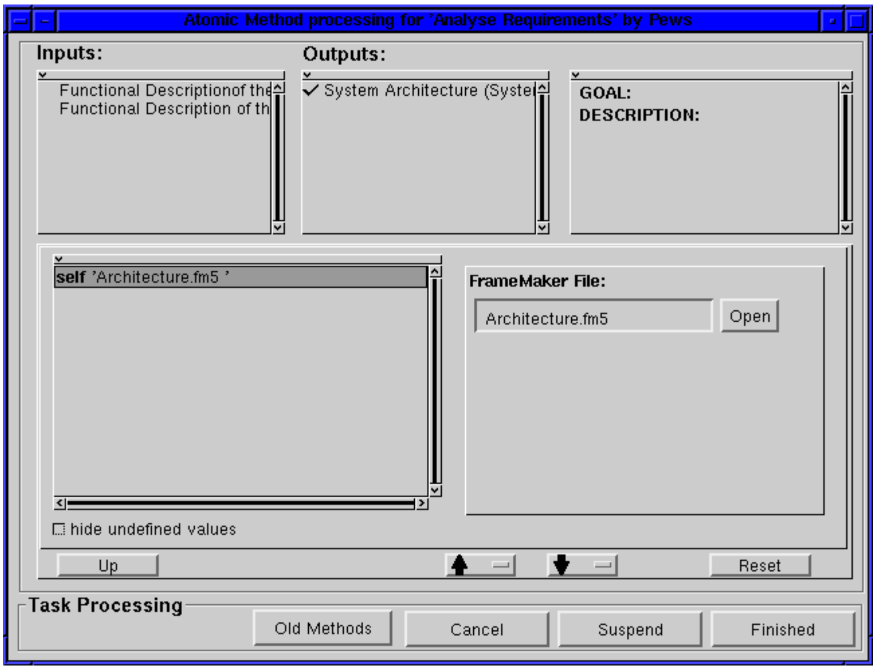

Figure 9 Creating products

has to be created, the agent can edit it in the lower part of the window. In this scenario, the product is a FrameMaker document with name „Architecture.fm5“. By pushing the „Open“, the document can be edited.

6.1 Recording the Plan

During project execution, the process model is successively instantiated. To clarify this and to separate the two levels of workflow description (i.e. model and instance), we use a different terminology. Furthermore, there is no one-to-one relation between the two levels, e. g. a process in the model can be instantiated several times. An instantiated process is called goal, an operator is the instantiation of a method and a variable is derived from a parameter.

To describe how the Scheduler executes a project plan, we have defined state transitions of the main instantiation elements, namely goals, operators and variables which are linked by decisions the agents made. A state transition is caused by an event. An event is triggered by an user or a state transition of a related instantiation element.

For each of those four elements, their states and possible transitions will be explained now in order to clarify how the Scheduler operationalizes a process model. For documentation purposes we use the OMT [37] notation.

Goals

A goal has two basic states: *valid* and *invalid*. *Valid* means that the goal is something to be worked on: one should search for a method to reach this goal. An *invalid* goal is currently not *valid*, but it may become so in the future. A transition from *valid* to *invalid* and vice versa is possible at any time.

Besides this two basic states, the state *valid* is split in several sub-states:

initialized
The initial state of a valid goal.

acceptable for planning
An agent can now create a plan how this goal can be reached, i. e. he may choose an operator for this goal. To become acceptable, two conditions must hold: the agents which are allowed to plan this goal have to be determined and the pre- and postconditions of the goal have to be satisfied. In other words it is possible to create a plan for this goal now.

accepted for planning
One of the agents has chosen to work on this goal, i. e. to try to find an operator for this goal.

performing
A (possibly different) agent is trying to apply an operator.

satisfied
An operator has been successfully applied.

unsatisfied
The operator could not be applied or an attempt to apply it failed.

The transitions for a *valid* goal are shown in Figure 10. The transition from the initial state *initialized* to *acceptable for planning* means that it is delegated for planning and its preconditions are satisfied. It will go back to the *initialized* state if the delegation is drawn back or if the preconditions are no longer satisfied.

Once a goal is *acceptable for planning*, an agent may pick this goal and start to choose an operator for it. The typical way for a goal is that if it is accepted to work on, first an operator will be chosen, then an agent will try to apply this operator. If he succeeds, the goal is *satisfied*. There are some exceptions to this. If the goal cannot be reached by applying the operator, the goal will be *unsatisfied*. Two things might have happened: Either the wrong operator was chosen or the agent was not able to apply the operator (e.g. was not familiar with a demanded technique). The corresponding transitions go to *accepted for planning* (in order to choose a different operator) or to *performing* (in order to retry the operator application). If no appropriate operator can be found in *accepted for planning*, the agent can give back the goal completely, it gets back into *acceptable for planning* again, so that another agent can accept it.

At some time in the future, the results of the chosen operator may be no longer suitable. It might become necessary to work again on this goal. In this case, the goal is set from *satisfied* to *acceptable for planning*.

Operators

An Operator has - similarly to a goal - two basic states *retracted* and *selected* which indicate that an operator is part of the current plan or not. An retracted operator may become selected again and a selected operator may be retracted at any time. The *selected* state is refined by a couple of sub-states:

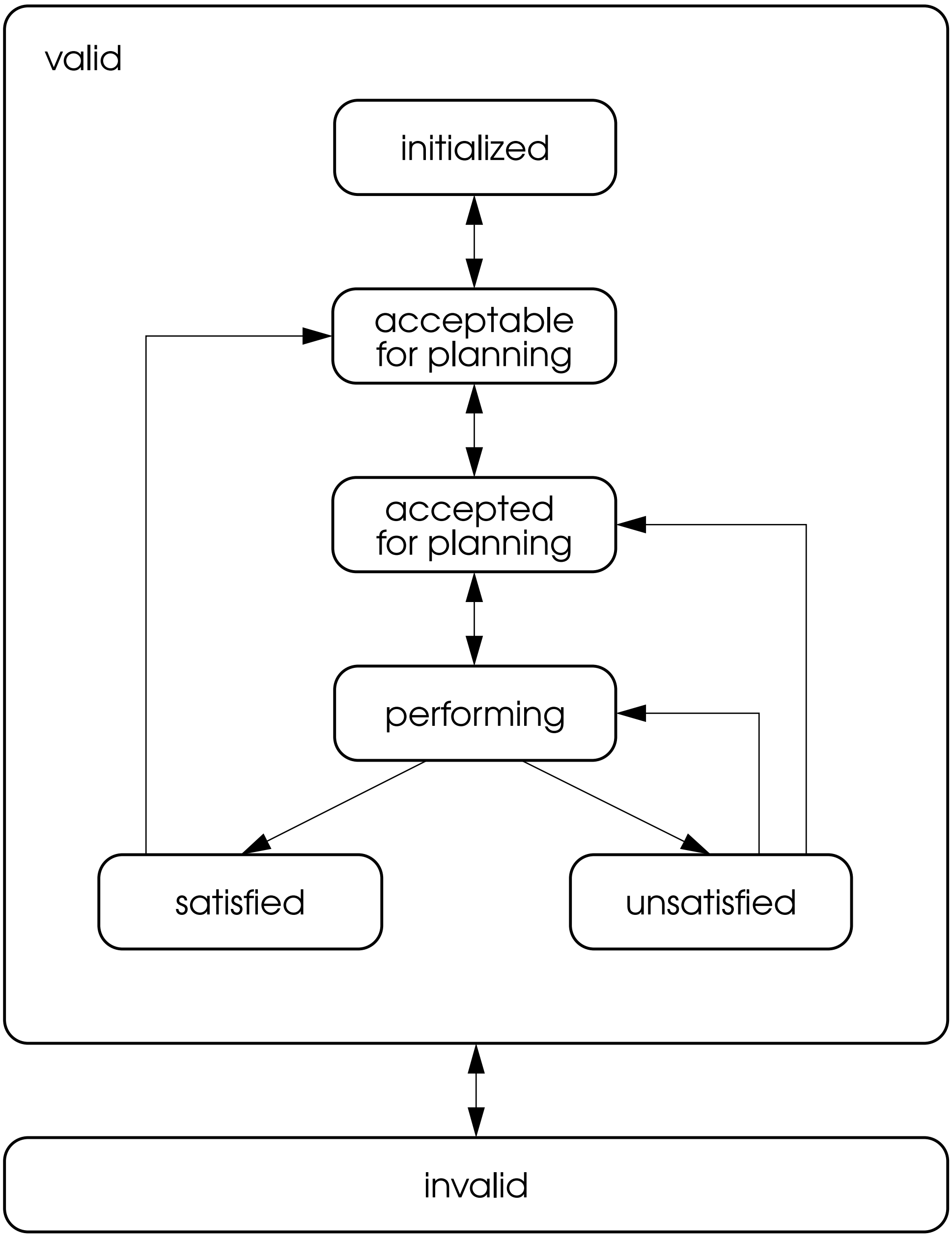

Figure 10 State transitions of a goal

initialized
The initial state of an operator.

acceptable for execution
The operator can be chosen by an agent, that means: It is determined which agents may try to apply this operator and the operator's preconditions are satisfied

accepted for execution

An agent has decided to execute the operator

execution started

An agent has started to execute the operator, i.e.he is working on it. The idea of this state is that operators which are being worked on should not be retracted as easy as operators in *acceptable for execution*. In that state, the agent has chosen to work on the operator but he has actually not spend any time for this job. If the operator is retracted, no work is lost.

execution finished

The agent's work has been finished. Now the postconditions have to be evaluated in order to check if the goal has been reached.

execution failed

The agent thinks that the goal cannot be reached using this operator.

execution succeeded

The goal has been reached using this operator.

Figure 11 shows the transitions for an operator: The first transition goes from *initialized* to *acceptable for execution*, as soon as the preconditions are satisfied and the operator is delegated.

After it has been marked as *acceptable for execution*, an agent can choose the operator for execution (*accepted for execution*) and start to work on it (*execution started)*. The point in time when an agent starts to work on an operator cannot be determined by the scheduler. This information has to be provided by the editor which is used by the agent, for example when an agent starts retrieving input information or begins typing. If an editor cannot provide this kind of information, the operator changes from *accepted for execution* to *execution started* immediately. When the agent considers his work to be done, he will confirm this, e. g. by pressing a button in the editor, and the operator's state will change to *execution finished*. After this - if the postconditions have been successfully checked - the operator moves to *execution succeeded*.

If the postconditions are not satisfied, the operator's state changes to *execution failed*. The agent now chooses what to do: Either it is a minor problem. Then he solves it and works again on the operator (*execution started*). Or he decides that he cannot solve the goal with this operator and changes its state to *acceptable for execution*. Now the agent who has chosen this particular operator is in charge of deciding what to do: One possibility is to leave the operator in *acceptable for execution*. This means that the goal can be achieved with this operator, but the executing agent was unable to do this. The other possibility is to choose a new operator which is represented by the state transition of the goal from *unsatisfied* to *accepted for planning* in Figure 10.

Decisions

A decision represents the selection of an operator or a delegation result. A decision can be *valid* or *retracted*. For example a valid decision for an operator means that the operator is part of the actual plan, i.e. it is *selected*. Also, rationales for decisions can be added. We distinguish between hard and weak rationales. Hard rationales for or against validity force the decision to become valid or retracted. Weak rationales will just be mentioned to the user, it is the user's task to change the status of his

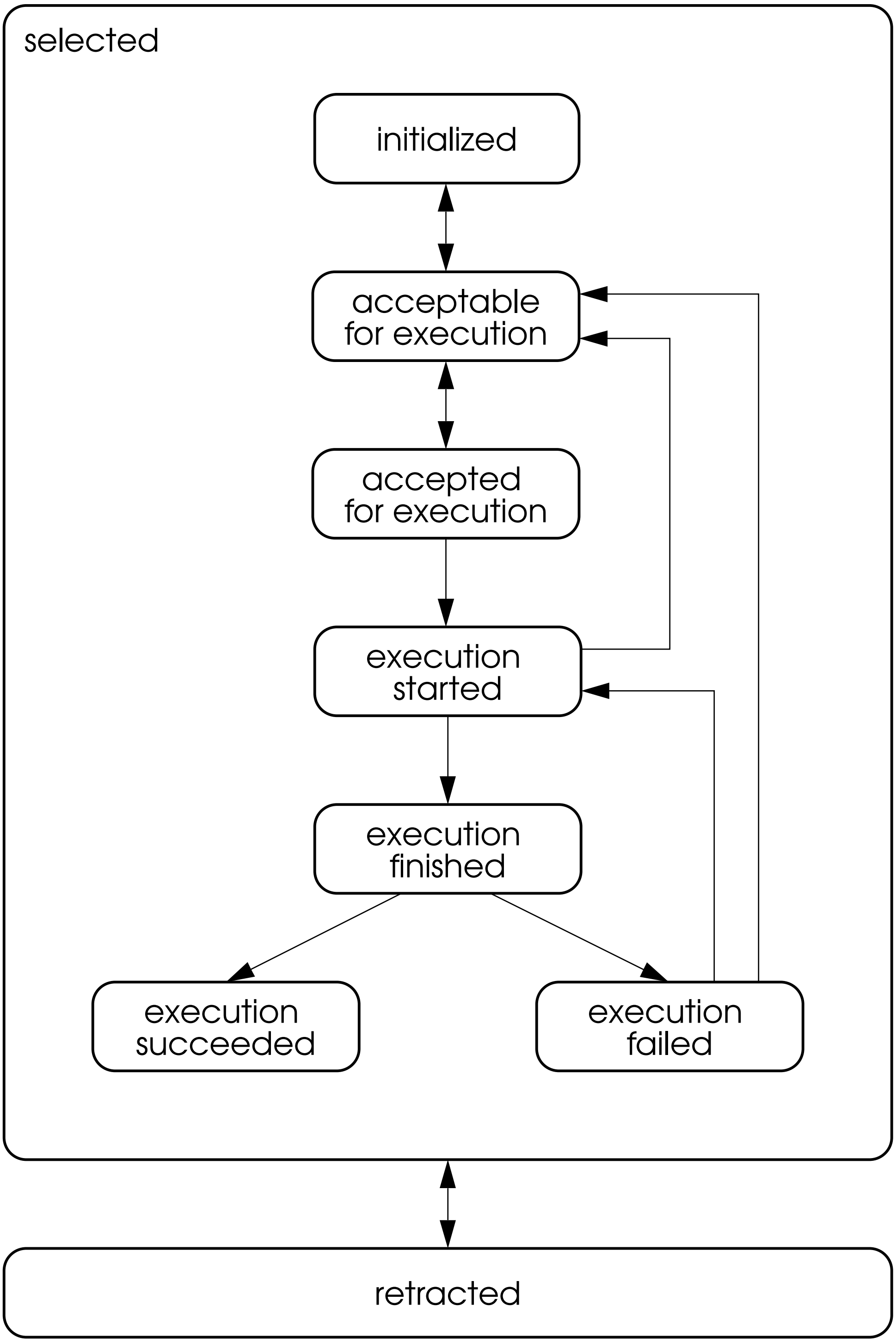

Figure 11 State transitions of an operator

decision. Weak rationales do not change the decision’s behavior and are not mentioned in the state diagram in Figure 12. The states are:

hard rationales for validity not existing
The decision is valid. There are no hard rationales for this, the decision can be retracted.

hard rationales for retraction not existing
The decision is retracted, but there are no hard rationales for this. It can become valid again.

hard rationales for validity existing
The decision is valid and cannot become invalid until the last rationale forcing it to be valid is removed.

hard rationales for retraction existing
The decision is forced to be retracted. It can only become valid if all hard rationales for this are removed

The Scheduler guarantees that there cannot be hard rationales for and against validity at the same time.

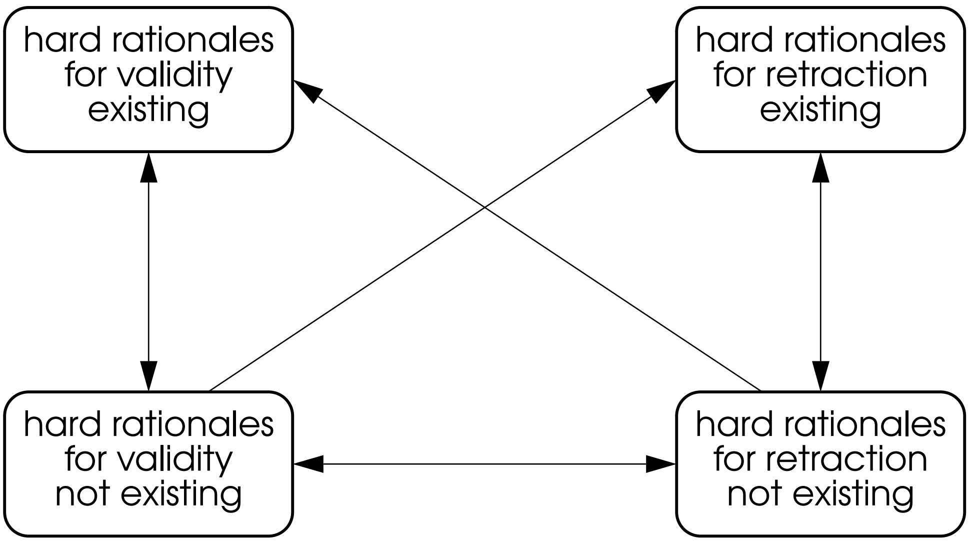

Figure 12 State transitions of a decision

Variables

A variable may be in one of the following states:

unassigned
The initial state of a variable. It has never been accessed by any user. Users which work on tasks depending on this variable should keep in mind that a value might been assigned to this variable in the future.

assigned
The variable holds a value.

irrelevant

An user has considered the variable not to be relevant for the current process. The variable has no value.

These three states have the transitions shown in Figure 13. It is possible to change from any state to another. The transition from *assigned* to itself represents the fact that the variable's value has been replaced by different one.

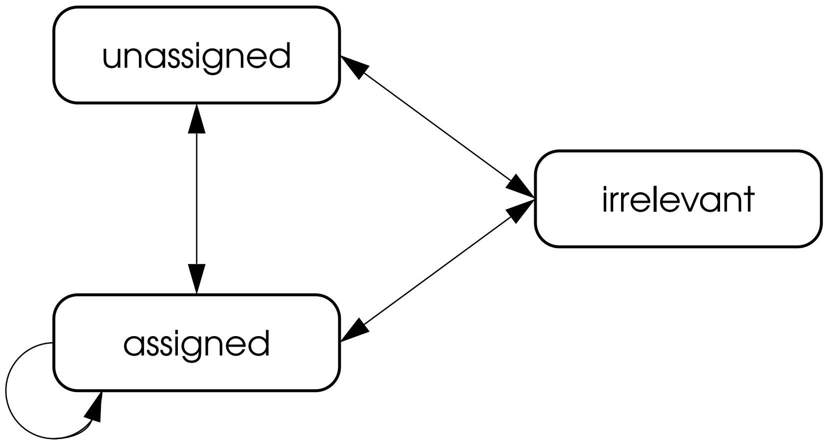

Figure 13 State transitions of a variable

6.2 Dependencies and Dependency Management

The Scheduler keeps track of the plan's execution, and manages the dependencies between the work several agents did. We distinguish between two kinds of dependencies: user defined dependencies and dependencies which are extracted from the process model. The dependency management mechanisms are discussed in detail in [10] and [29].

6.2.1 Representation of Dependencies

To record the agents' activities, a dependency network is constructed. Dependencies are established between *decisions, goals,* and *variable assignments*. They are formulated as logical implications. This representation of the plan's execution is simple enough to generate a record automatically, and still powerful enough to support change management. Within the state transition diagrams of Section 6.1, dependencies occur as conditions for transitions. As soon as a decision or assignment changes its basic state (e.g. a decision is retracted), it is easy to determine all affected assignments and decisions and to calculate the effects of this change. We employ an extension of the design model REDUX [34] for this task. REDUX tracks dependencies resulting from process decomposition. CoMo-Kit as well as REDUX use a Truth Maintenance System (JTMS [11]) known from Artificial Intelligence to manage dependencies and to handle the effects of changes efficiently.

6.2.2 Automatic Derivation of Dependencies

Dependencies are automatically generated from the relations specified in the plan:

Goals and Subgoals: One criteria for the validity of a decision is the validity of the goal it belongs to. The validity of a subgoal is justified by the validity of the decision for the operator of its parent goal. As soon as the parent decision is retracted, the subgoals become invalid, and as a result, the child decisions are retracted, too. If the parent decision becomes valid again, the decision for its children are re-validated, too.

Product and Parts: A similar relation like the one above exists along the part-of hierarchy of products. Here, the validity of assignments for subparts depend on the validity of their parent-products.

Input-Output: The most important dependency derives from the following assumption: An agent who made a decision using some information and knowledge about certain other assignments and decisions might change his decision when the data has been changed. Therefore, justifications from all input-assignments to the decision are generated based on the product flow of the associated method.

6.2.3 User defined dependencies

In addition an agent can edit dependencies manually: he/she can add new dependencies, modify or remove self defined or automatically generated dependencies, in order to be notified more or less often.

7 Increasing flexibility of planning and execution

With the presented terms of the ontology, explicit project plans can be created. These plans describe the general course of action of projects.

In our approach an initial plan can be further specified during execution. Delaying parts of the plan specification until execution has the advantage that specific project knowledge such as products, produced by preceding activities, rationales for or against decisions, and resource assignments, can be used to complete and adapt the plan.

Furthermore, we provide basic mechanisms to change project plans as a reaction to changing conditions or plan errors. By managing causal dependencies, the effects of changes can be computed and handled.

The following sections describe in more detail how our techniques lead to more flexible planning and execution.

7.1 Refining the Plan

One goal of our approach is to allow both the planner and the project manager to delay planning decisions until execution, when they have access to project specific information. Such decisions are

- *operator selection.* The selection of an applicable operator to solve a goal in the state *accepted for planning* causes a state change of both the goal and the operator: the operator changes to *initialized*, the goal to *satisfied.* For decision support, data from preceding activities can be used. For example, a manager is responsible for planning the test phase of implemented software. Input for this activity is the program code. The model provides two testing alternatives, „equivalence class based testing“ and „code reading“ (see Figure 4). Because the given code would generate many different equivalence classes to be tested, the manager decides in favor of code reading.

- *delegation of goals for planning activities.* A goal in the state *initialized* has to be delegated to one or more planners. The delegation activity triggers delegation events for each person the goal is delegated to. If additionally all *planning preconditions* are satisfied, the goal is *acceptable for planning*. The benefit of this late binding is that concrete resource assignments can be made, depending on the current situation.
- *delegation of operators for execution activities.* Planning a goal includes the delegation of the selected operators to appropriate agents. For this, the planning agent triggers a delegation event for every selected agent. If one or more delegation events occur, and the *execution preconditions* are satisfied, the operator changes to *acceptable for execution*. Now each of the agents, who the goal has been delegated to can accept the operator.

Sometimes it is desirable to predefine the general course of action and to delay parts of the modeling until execution: In doing so, project specific knowledge can be used for detailed modeling. We allow to extend the process models in two ways:

- *Adding new methods to the model.* During execution, increasing process knowledge may result in new solution approaches. The agent can add new methods to the process model. The new method is immediately available to the current project. For example, an agent defines a new method to create test cases for a software program. He/she extends the process „Implementation process“ of the example in Section 5.3, Figure 4 by a new method named „Implementation with Boundary Value Analysis“ (see marked parts of Figure 14).

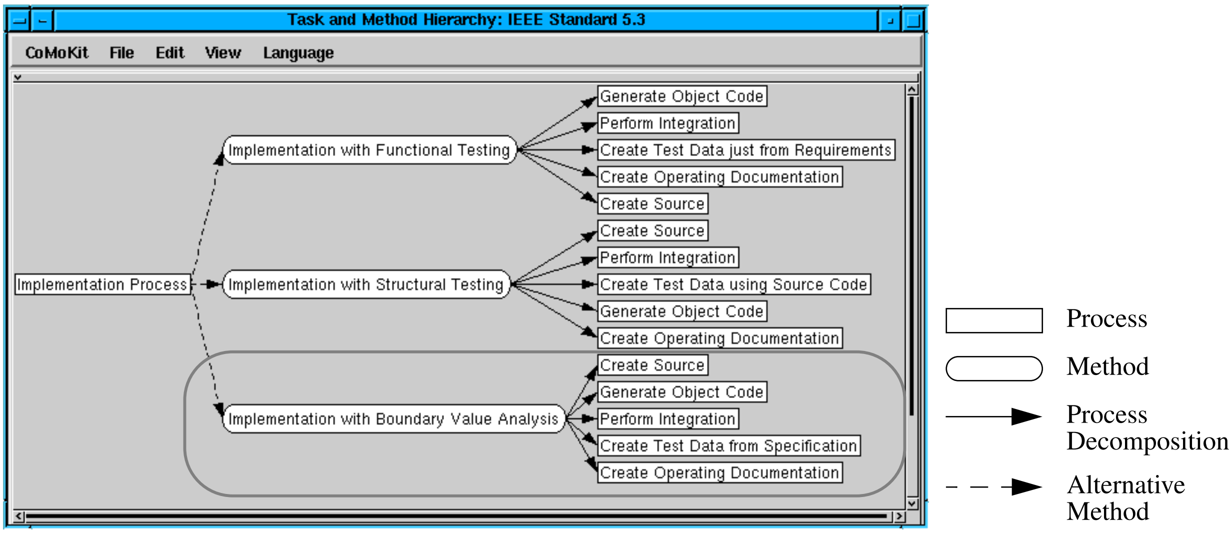

Figure 14 Extended decomposition graph of process „Implementation Process“

- *Refining methods.* Many methods can be planned in detail only on basis of project specific knowledge. Those methods stay unspecified in the initial model. The „architectural design“ of a software product, for example, cannot be planned before the „requirements document“ has been produced.
  As soon as the required information becomes available, the refinement of the method can be completed. Because the scheduler manages product flow dependencies, it informs the responsible

planner that the required information is available. Now, the project planner extends the method specification in the process model by adding new processes, specifying the product flow between them, and refining the subprocesses within the method definition using the available product data. The model changes are propagated to the Scheduler, which in response updates its internal state.

- *Adding new qualifications.* Qualifications to solve a goal as well additional skills of the agents can be added to the model. The Scheduler analyses the new qualifications and computes the set of agents, who are allowed to solve the goal.
- *Adding new resources.* Changing conditions may force to include new resources in the project. Firstly, the new resource has to be added to the model. Secondly, resource specific skills have to be assigned to it.

7.2 Changing the plan

During project execution changing conditions and planning errors force the users to discard solutions. These changes affect the project plan as well as the produced products. As a result, the plan has to be adapted to the new situation. In such situations, the benefit of managing dependencies between events and plan states becomes visible: the effects of plan changes are handled by the system, affected team members can automatically be notified, and are guided in to react in an appropriate way. Our approach provides the following mechanisms to react to events external to the system, or to state changes within the plan.

7.2.1 Changing decisions

As described in the previous section, the project members can select operators and delegate agents during execution. If needed, these decisions can be rejected:

- *Rejecting planning decisions.* The decision for an operator can be rejected by sending a *retract* event to the selected operator. As a result the affected operator and the corresponding goal change their states: The operator changes to *execution failed*, the goal returns to *acceptable for planning*. Such changes can happen on abstract planning levels as well on a fine granular level. With the managed dependencies, affected parts of the project model can be computed and state changing events of dependent goals and operators triggered. Later on the planner can decide for another fitting operator.
- *Rejecting delegations*. Rejecting a delegation may be necessary if time and personnel conditions change (for example an employee leaves the enterprise during project execution). Such events have no consequences if the goal is already in execution by another agent.

7.2.2 Changing the process model

In case of errors in the model, rejecting a decision is insufficient, and the process model has to be corrected. Unfortunately, modeling errors are often not noticed before the plan is already in execution. Nevertheless, the errors have to be corrected to guarantee a correct execution. Therefore, the system has to provide mechanisms to propagate model changes to the plan in execution, while saving as much plan knowledge as possible. Errors can be eliminated within

- *method & process definitions.* We allow to correct errors within method or process definitions e.g. to add or remove pre- and postconditions, to remove methods or to change the product flow. For example, the product flow between two processes „architectural design“ and „implementation“ may have the wrong direction: the process “architectural design“ wrongly consumes the already implemented program. To guarantee a correct execution, the project planner corrects the product flow within the model before he selects the method.

- *agent bindings.* Skills of agents may change, as well as skills that are necessary to work on a task. In both cases, the project manager can adapt the corresponding parts of the model.

8 State of Implementation

The CoMo-Kit Modeler as well as the CoMo-Kit Scheduler have been prototypically implemented for a local area network. The integration of CoMo-Kit with the World Wide Web, using the Visualwave package, currently allows for plan execution and method selection/rejection [8]. Defining new methods currently is not supported via the Web.

In Figure 15 the architecture of the CoMo-Kit Scheduler is shown.

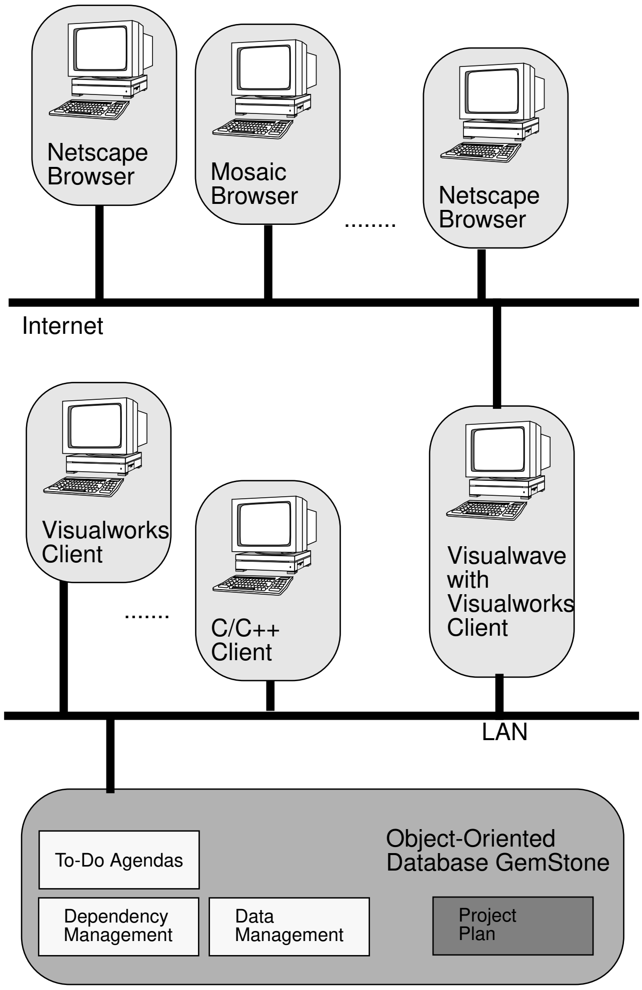

Figure 15 Architecture of the Scheduler

The server component of the Scheduler is implemented with the object-oriented database management system GemStone by Servio Corp. The server

- stores the current project model,
- handles the worklists for every agent,

- stores all data produced during process execution[4], and
- manages dependencies between project information.

The server component is accessible via a local area network from Visualworks for Smalltalk-80 and C/C++-written clients. We developed clients in Visualworks which allow to
- accept processes to work on
- plan a process
- change plans
- decompose processes into finer-grained subprocesses
- supervise how the work on the subprocesses is advancing
- edit products

For every task, a separate client can be used for planning. Therefore it is possible to distribute the overall planning process to several agents via computer networks. Clearly, it is also possible to distribute the overall workflow (i.e. each of a process's subprocess) to several agents.

9 Related Work

In the following, we discuss related work in the area of workflow management and software environments according to flexibility.

Within workflow management research, many publications emphasize the importance of flexible workflow modeling and enactment. Unfortunately, concrete solution approaches to the above problems are rarely described [4, 9, 31, 11, 23], and lack an integration of dependency management mechanisms.

A research field with more concrete solutions for this problem domain is software engineering. Main approaches that give special attention to flexibility aspects are discussed here.

Marvel [25] follows a rule-based approach to express assumptions for the enactment of process steps. The approach supports forward-chaining of rules as well as backward reasoning. In Marvel decisions are not explicitly represented. Therefore it is not possible to express causal dependencies between decisions and products. Mechanisms for changing decisions are not discussed.

The MERLIN [24] approach has some similarities to Marvel. In MERLIN the process engine is some kind of inference machine that works on facts, specifying the (current state of the) software project. Activities performed on documents change their content and state. Dependencies between the state of documents are formulated as rules and evaluated within the process engine. Extending the set of rules and facts during execution seems to be possible. As in Marvel, decisions are not represented. Mechanisms to allow for changing the solution method and handling the resulting effects are not discussed.

GRAPPLE [20] is based on an explicit model of planning. Plan are constructed dynamically by instantiating operators. In GRAPPLE it is assumed that, for planning and plan recognition, knowledge is needed which is not included in operator definitions. This knowledge is given as a set of assumptions

4. If the data is produced using an external tool (e.g. Framemaker, CAD Systems etc.) then only a reference to the file is stored.

and handled by a Reason Maintenance System. This knowledge is mainly used to constrain the set of applicable operators. Causal dependencies between decisions are not represented.

The SPADE [2] environment provides a process language called SLANG to support enactment and dynamic evolution of a process model. With its reflexive nature, it allows the process engineer to define the behavior of the process engine himself/herself. In contrast to our approach, the specification of concrete execution and change strategies is left to the user. As a result, the SPADE system does not incorporate automated strategies which handle the effects of changes.

As in our approach EPOS [26] provides a set of predefined concepts to model project plans. Changing preconditions, postconditions and the task decomposition of a plan (automatically generated by an AI planner) is possible. If such changes occur, the AI planner replans those parts of the plan that are affected by the changes. In contrast to our approach, replanning is done automatically. The user is not involved in the planning and replanning activities. It is not discussed what happens with those parts of the plan that already have been executed nor with the crated products. No mechanisms to notify affected team members are proposed.

Currently, CoMo-Kit manages decisions made during a project but does not handle constraints which restrict the possible alternatives for a decision. We work on the integration of a constraint problem solver to incorporate this functionality. In [19], for example, a similar approach is described.

For practical reasons, an integration of our workflow management techniques with standard project management tools (e.g. MS Project) is important. [7] describes another approach to this problem, although it is more restricted than CoMo-Kit in terms of change management and replanning.

10 Discussion & Future Work

The CoMo-Kit approach increases the flexibility of computer-supported work processes by applying knowledge-based techniques. The actual plan guides team members in their daily work and is the basis from which traceability can be reached. Hence, CoMo-Kit has similar advantages as current workflow approaches. Alternating planning and execution steps provides the flexibility of groupware systems without their disadvantages.

CoMo-Kit can be used for project coordination [27]. The system notifies appropriate team members after a change has happened. Hence, it reduces the risk that the outcome of a project is incorrect.

An extension of CoMo-Kit will deal with project scheduling. For this, we will adopt the techniques developed by Sigrid Goldmann at Stanford University [18].

The CoMo-Kit Scheduler, from an abstract point of view, contains a logically central component which is used by all team members and is responsible for worklist management and change propagation. We will adopt an agent-oriented approach, as for example in [34] or [1], to distribute the functionality over Local and Wide Area Networks.

A long term goal is the reuse of „good“ old project plans for new tasks. This requires an „Experience Factory“ [3] and efficient retrieval and adaptation techniques. It is a basis for long-term process improvements resulting in increased competition power for the enterprise.

11 References

[1] M. Barbuceanu, Mark S. Fox, Coordinating Multiple Agents in the Supply Chain, in: Proc. WET ICE 96, (IEEE Computer Society Press, U. S., 1996), 134 -141.

[2] Sergio C. Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi. Software process model evolution in the SPADE environment, in: *IEEE Transactions on Software Engineering*, 19(12), December 1993, 1128–1144.

[3] V. R. Basili: The Experience Factory and its Relationship to Other Improvement Paradigms, in: Ian Sommerville, Manfred Paul, eds., Proc. of the 4th European Software Engineering Conference, Lecture Notes in Computer Science Nr. 706 (Springer Verlag, 1993), 68-83.

[4] Douglas. P. Bogia, Simon. M. Kaplan: Flexibility and Control for Dynamic Workflows in the wOrlds Environment, in: Proceedings of the Conference on Organizational Computing Systems, (ACM Inc., 1995).

[5] J. Breuker, W. van de Velde, eds.: CommonKADS Library for Expertise Modeling, (IOS Press, 1994).

[6] A. Bröckers, Ch. M. Lott, H. D. Rombach, and M. Verlage, MVP–L language report version 2. Technical Report 265/95, (Department of Computer Science, University of Kaiserslautern, 1995).

[7] K.J. Cleetus, C. Cascaval, K. Matsuaki, PACT - A Software Package to Manage Projects and Coordinate People, in: Proc. WET ICE 96, (IEEE Computer Society Press, U. S., 1996). 162-169

[8] CoMo-Kit: On-line Web demo, http://wwwagr.informatik.uni-kl.de/~comokit/www-interface.html

[9] W. Deiters, V. Gruhn, R. Striemer: Der FUNSOFT-Ansatz zum integrierten Geschäftsprozeflmanagement, in: Wirtschaftsinformatik, 37, (1995).

[10] B. Dellen, K. Kohler, F. Maurer, Integrating Software Process Models and Design Rationales, in: Proceedings of the eleventh Knowledge Bases Software Engineering Conference KBSE 96, (IEEE Computer Society, U. S.), 84- 93.

[11] Gert Florijn, Timo Besamusca, Dany Greefhorst: Ariadne and HOPLa: Flexible coordination of collaborative processes, in Proceedings of Coordination'96, Cesena, Italy, Springer Verlag, 1996).

[12] A. Fuggetta, A Classification of CASE Technology, in: Computer, Vol. 26, (1993).

[13] J. Galler, J. Hagemeyer, A.-W. Scheer, The Coordination of Interdisciplinary Teams in Workflow Projects, in: Proc. IDIMT-95, (Kubova Hut, Czech Rebublic)
http://www.iwi.uni-sb.de/forschungsprojekte/contact/cont_2in.html

[14] J. Galler, A.-W Scheer, Workflow-Projekte: Vom Geschäftsprozeßmodell zur unternehmensspezifischen Workflow-Anwendung, in: A.-W Scheer, ed., IM-Information Management 10 1 (1995), 20-28
http://www.iwi.uni-sb.de/forschungsprojekte/contact/cont_4in.html

[15] Pankaj K. Garg, Mehdi Jazayeri: Process centered software engineering environments (IEEE Computer Soc. Pr. , 1996. - XII, ISBN 0-8186-7103-3).

[16] D. Georgakopolous, M. Hornick, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, in: Distributesd and Parallel Databases 3, (1995) 119-153.

[17] W. W. Gibbs, Software's chronics crisis , in: Aci. Am. (1994) 86-95.

[18] S. Goldmann, Procura: A Project Management Model of Concurrent Planning and Design, in: Proc. WET ICE 96, (IEEE Computer Society Press, U. S., 1996). 177-183

[19] L. Gupta, J. Chionglo, M. Fox, A Constraint-Based Model of Communication and Coordination in Concurrent Design Projects, in: Proc. WET ICE 96, (IEEE Computer Society Press, U. S., 1996).

[20] k. E. Huff, V. R. Lesser: An Plan-based Intelligent Assistant That Supports the Software Development Process, in: Proceedings of the Third Symposium on Software Development Environments (ACM Inc., 1988).

[21] Humphrey, S. Watt, Recent findings in software process maturity, in: A. Endres and H. Weber, eds, Software Development Environments and Case Technology, Lect. Notes Computer. Sci., No. 509, (Springer Verlag, Berlin, 1991) 258-270.

[22] Humphrey, S. Watt, Managing the Software Process, SEI Series in Software Engineering, (Addison-Wesley Publishing Company, Inc.)

[23] S. Jablonski: MOBILE: a Modular Workflow Model and Architecture, in: Proceedings of the Fourth International Working Conference on Dynamic Modelling and Information Systems, (1994).

[24] G. Junkermann, B. Peuschel, W. Schäfer, S. Wolrf: Merlin: Supporting Cooperation in Software Development through a knowledge based Environment, in: A. Finkelstein, J. KRamer, B. Niseibeh (editors), Software Process Modelling and Technology (Research Studies Press, UK, 1994).

[25] G.E. Kaiser P.H. Feiler, S.S. Popovich: Intelligent Assistance for Software Development and Maintenance (*IEEE Software*, May 1988).

[26] M. Letizia Jaccheri and Reidar Conradi. Techniques for process model evolution in EPOS, *IEEE* Transactions on Software Engineering, 19(12):1145–1156 (IEEE Computer Society Press, December 1993).

[27] F. Maurer, Computer Support for Project Coordination (Workshop Summary), in: Proc. WET ICE 96, (IEEE Computer Society Press, U. S., 1996), http://wwwagr.informatik.uni-kl.de/~maurer/WETICE96_Papers/summary.html, 200-205.

[28] F. Maurer, G. Pews: Supporting Cooperative Work in Urban Land-Use Planning, in: Proc. COOP-96, (INRIA, Sophia Antipolis,1996) 663-679.

[29] F. Maurer, J. Paulokat: Operationalizing Conceptual Models Based on a Model of Dependencies, in: A. Cohn, ed., ECAI 94. (John Wiley & Sons, Ltd., 1994)

[30] M. Imai, Kaizen (Wirtschaftsverlag Langen Müller Herbig, 1992)

[31] A. Oberweis: Modellierung und Ausführung von Workflows mit Petri-Netzen. Teubner-Reihe Wirtschaftinformatik (Teubner Verlag, 1994).

[32] Ch. Petrie: Planning and Replanning with Reason Maintenance, Dissertation, University of Texas, Austin, 1991.

[33] Ch. Petrie, M. Cutkosky. Design space navigation as a collaborative aid, in: J. Gero and F. Sundweeks, ed., Artificial Intelligence in Design ‘94 (Kluwer Academic Publishers, 1994).

[34] Ch. Petrie: The Redux' Server, Proc. ICICIS, (Rotterdam, 1993).

[35] Proceedings Workshops on Enabling Technologies: Infrastructure for Collaborating Enterprises, (IEEE Computer Society Press, 1996).

[36] H. D. Rombach, M. Verlage, Directions in Software Process Research, in: M. V. Zelkowitz, ed., Advances in Computers, Volume 41 (Academic Press, Boston, 1995) 1-63.

[37] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object Oriented Modeling and Design, (Prentice-Hall, Inc. , 1991).

[38] M.Verlage, Multi–view modeling of software processes, in: B. C. Warboys, ed., Proc. Third European Workshop on Software Process Technology, (Springer–Verlag, 1994) 123–127.

[39] M. Verlage, B. Dellen, F. Maurer, J. Münch, A Synthesis of two Process Support Approaches, in: Proceedings of the 8th Software Engineering and Knowledge Engineering Conference, SEKE'96 (Knowledge Engineering Institute, June 1996).

[40] Workflow Management Coalition: Terminology & Glossary, http://www.aiai.ed.ac.uk/WfMC/DOCS/glossary/glossary.html

[41] J. Wainer, M. Weske, G. Vossen , C. M. Bauzer Medeiros, Scientific workflow systems, in: Proc. NSF Workshop on Workflow and Process Automation, http://wwwmath.uni-muenster.de/~dbis/Weske/papers/wwvm96.html

Barbara Dellen received her Diploma in Computer Science from the University of Kaiserlautern and works as researcher in the SFB 501 „Development of large systems with generic methods“ located at the University of Kaiserslautern. The focus of her research is dynamic process planning, especially in Software Engineering, dependency management, and design rationales.

Frank Maurer is a senior researcher in the expert system group of the University of Kaiserslautern. His interests include design processes, software engineering, workflow management, distributed knowledge-based systems, and case-based reasoning. Current projects include the CoMo-Kit, which supports the modeling and management of complex work processes. He received a Ph.D. in Computer Science at the University of Kaiserslautern in 1993 in the area of Hypermedia & Knowledge Engineering for distributed knowledge-based systems. He is a member of the editorial board of the IEEE Internet Computing magazine and was a program committee member for several national and international conferences and workshops.

Gerd Pews currently works on integrating knowledge-based techniques and workflow management for urban land-use planning. His main research interests are Knowledge Engineering, distributed knowledge-based systems, case-based reasoning, and object-oriented programming. In 1994, he received his Diploma in Computer Science from the University of Kaiserslautern.